Feature

High available application | No downtime

Frequent deployments

Opensource

By google

Kubernetes is a production-grade, open-source platform that orchestrates the placement (scheduling) and execution of application containers within and across computer clusters.

Kubernetes Clusters

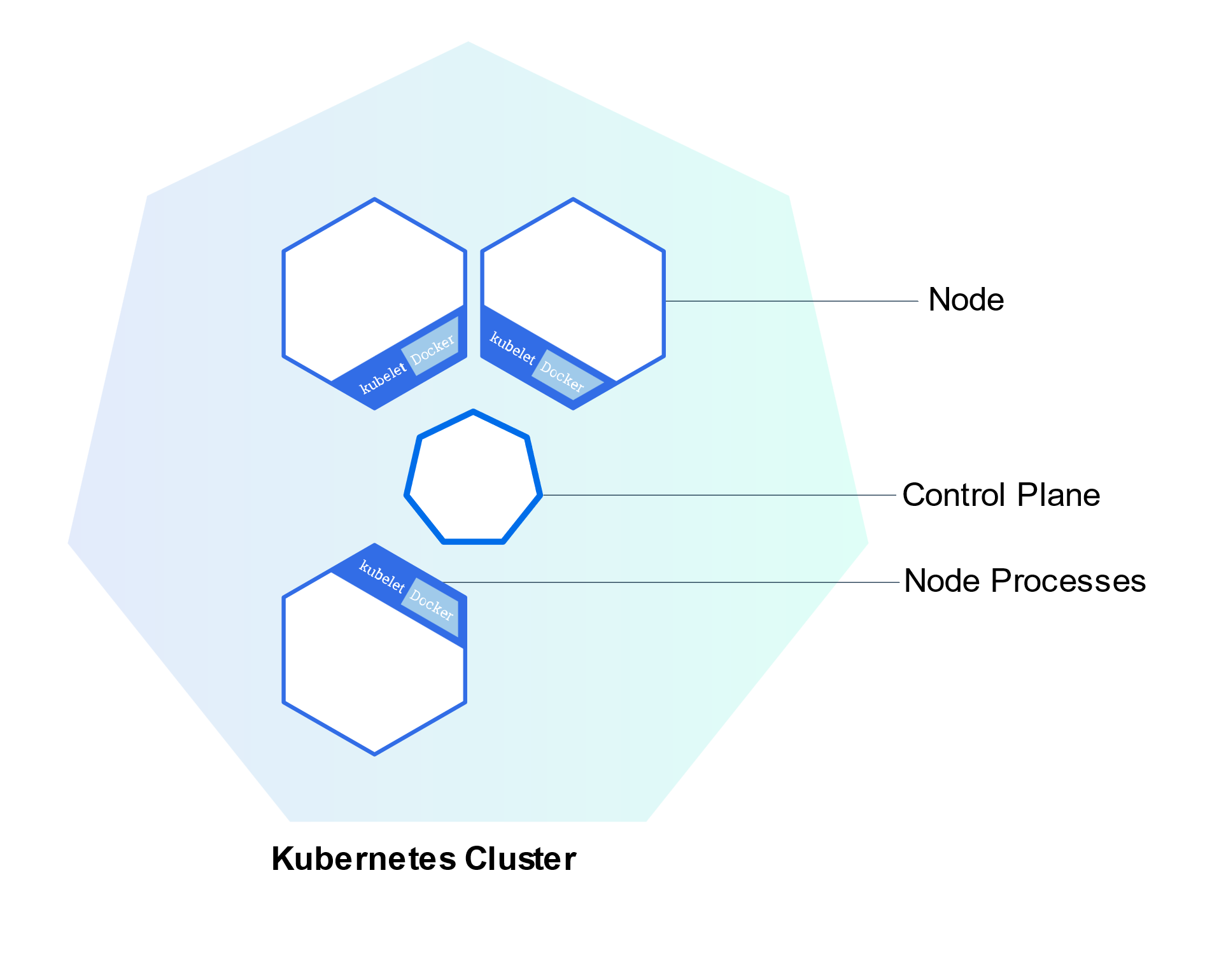
Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.

Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.

**A Kubernetes cluster consists of two types of resources:**

1. The Control Plane coordinates the cluster
2. Nodes are the workers that run applications

Cluster Diagram



Control Planes

Control Planes manage the cluster and the nodes that are used to host the running applications.

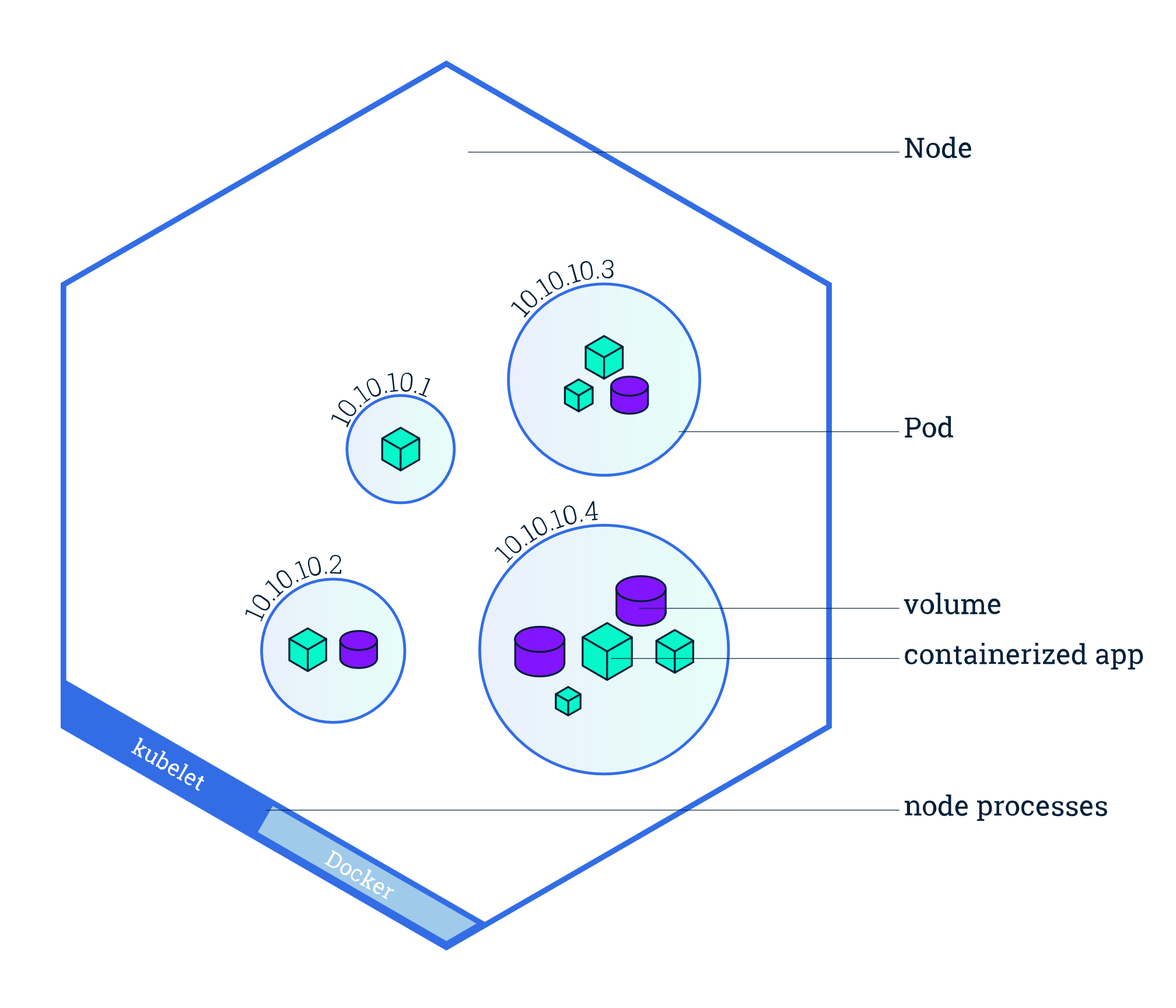
The Control Plane coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

**The nodes communicate with the control plane using the**[**Kubernetes API**](https://kubernetes.io/docs/concepts/overview/kubernetes-api/), which the control plane exposes. End users can also use the Kubernetes API directly to interact with the cluster.

Node/ Worker Machine

A node is a worker machine in Kubernetes and may be a VM or physical machine, depending on the cluster. Multiple Pods can run on one Node.

* Kubelet, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.
* A container runtime (like Docker) responsible for pulling the container image from a registry, unpacking the container, and running the application



Minikube

Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node.

minikube version = to get the version of minikube

minikube start = Minikube started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

kubectl version = provide both the version of the client and as well as the server. The client version is the kubectl version; the server version is the Kubernetes version installed on the master.

kubectl cluster-info = To get the cluster details

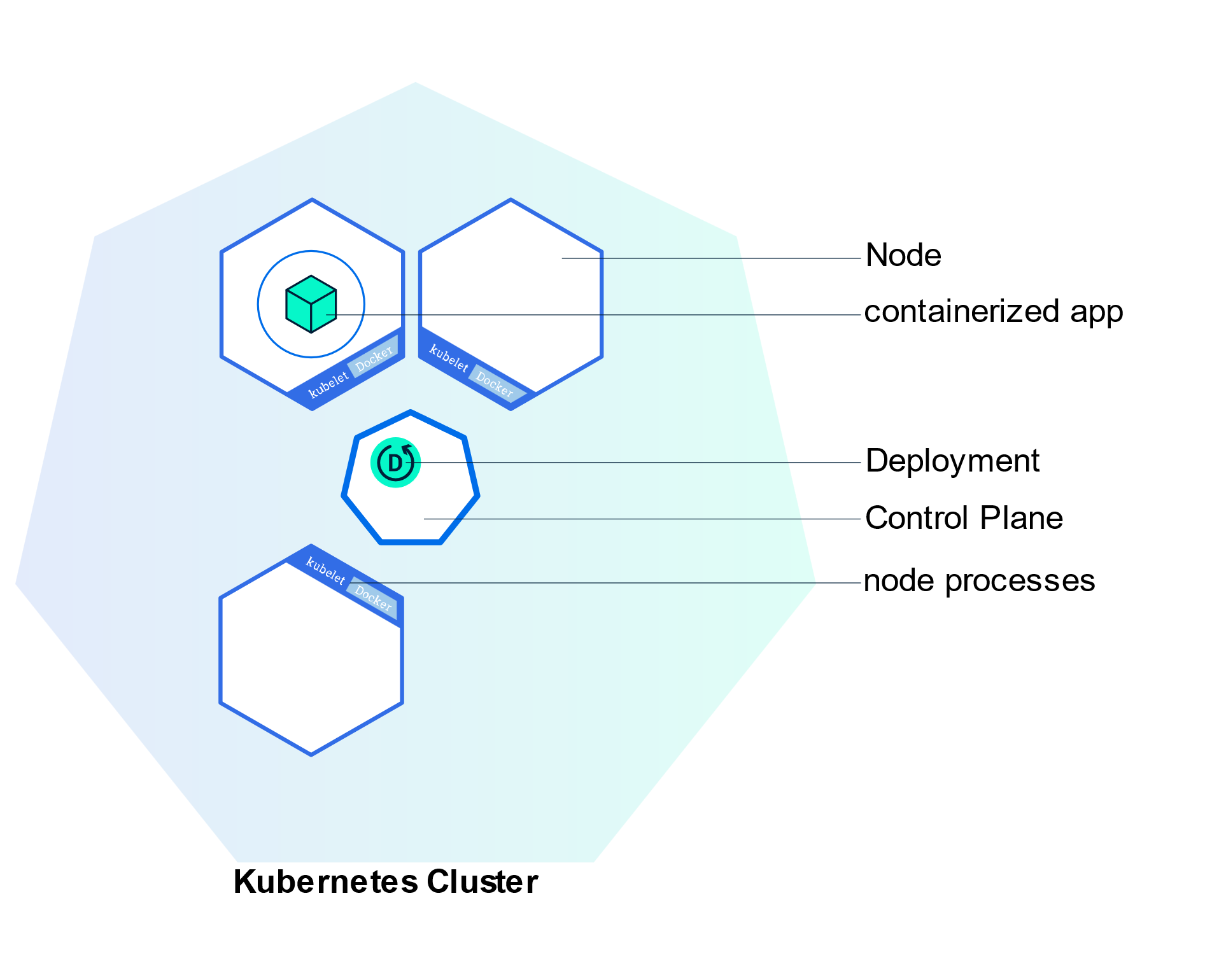
kubectl get nodes = This command shows all nodes that can be used to host our applications. And status is ready (it is ready to accept applications for deployment).

Deployment

A Deployment is responsible for creating and updating instances of your application.

Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**

# Deploying your first app on Kubernetes



Applications need to be packaged into one of the supported container formats in order to be deployed on Kubernetes

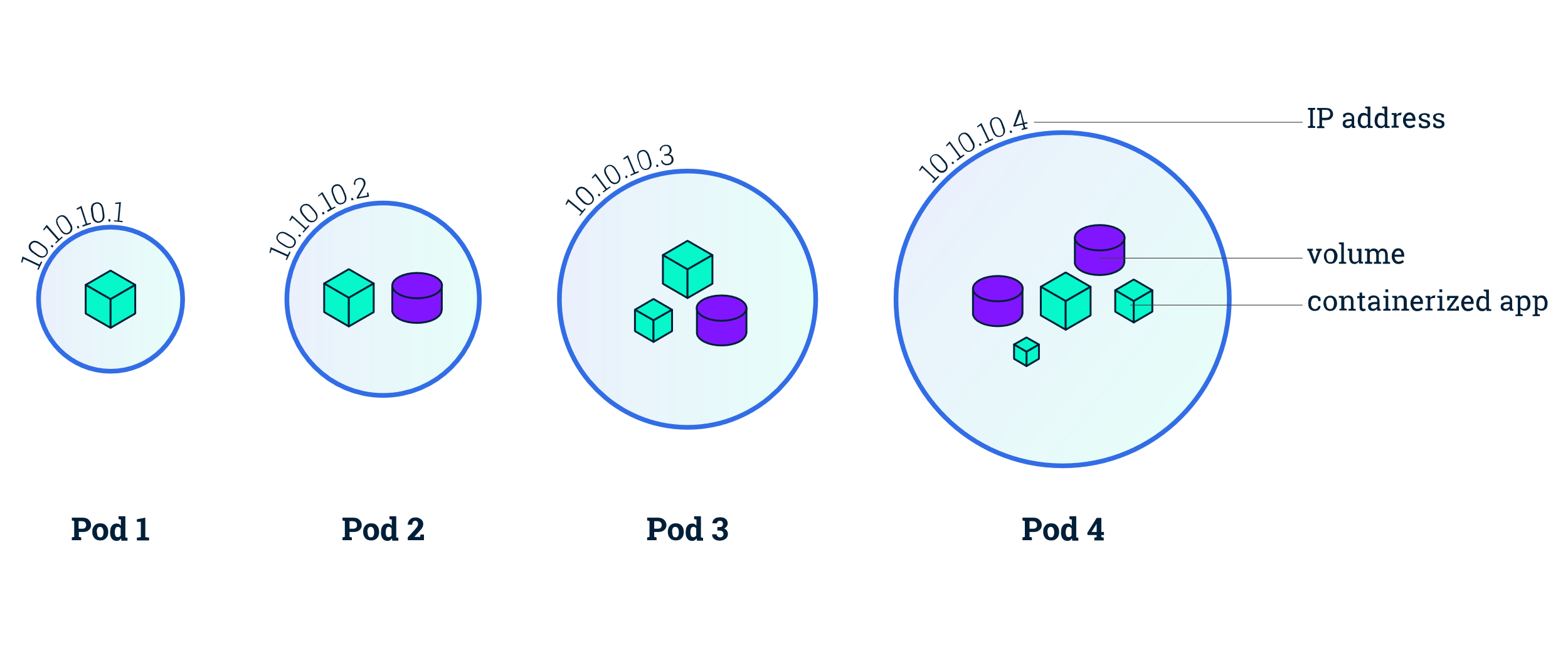
kubectl create deployment <deploymentName> --image=<imagePath> = need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker hub).

Pod

A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker), and some shared resources for those containers. Those resources include:

* Shared storage, as Volumes
* Networking, as a unique cluster IP address
* Information about how to run each container, such as the container image version or specific ports to use

Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.



Containers should only be scheduled together in a single Pod if they are tightly coupled and need to share resources such as disk.

kubectl describe pods :The view what containers are inside that Pod and what images are used to build those containers. **The describe command can be used to get detailed information about most of the kubernetes primitives: node, pods, deployments.**

Anything that the application would normally send to STDOUT becomes logs for the container within the Pod. We can retrieve these logs using the kubectl logs command

execute commands directly on the container once the Pod is up and running. For this, we use the exec command and use the name of the Pod as a parameter.

Service

A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.

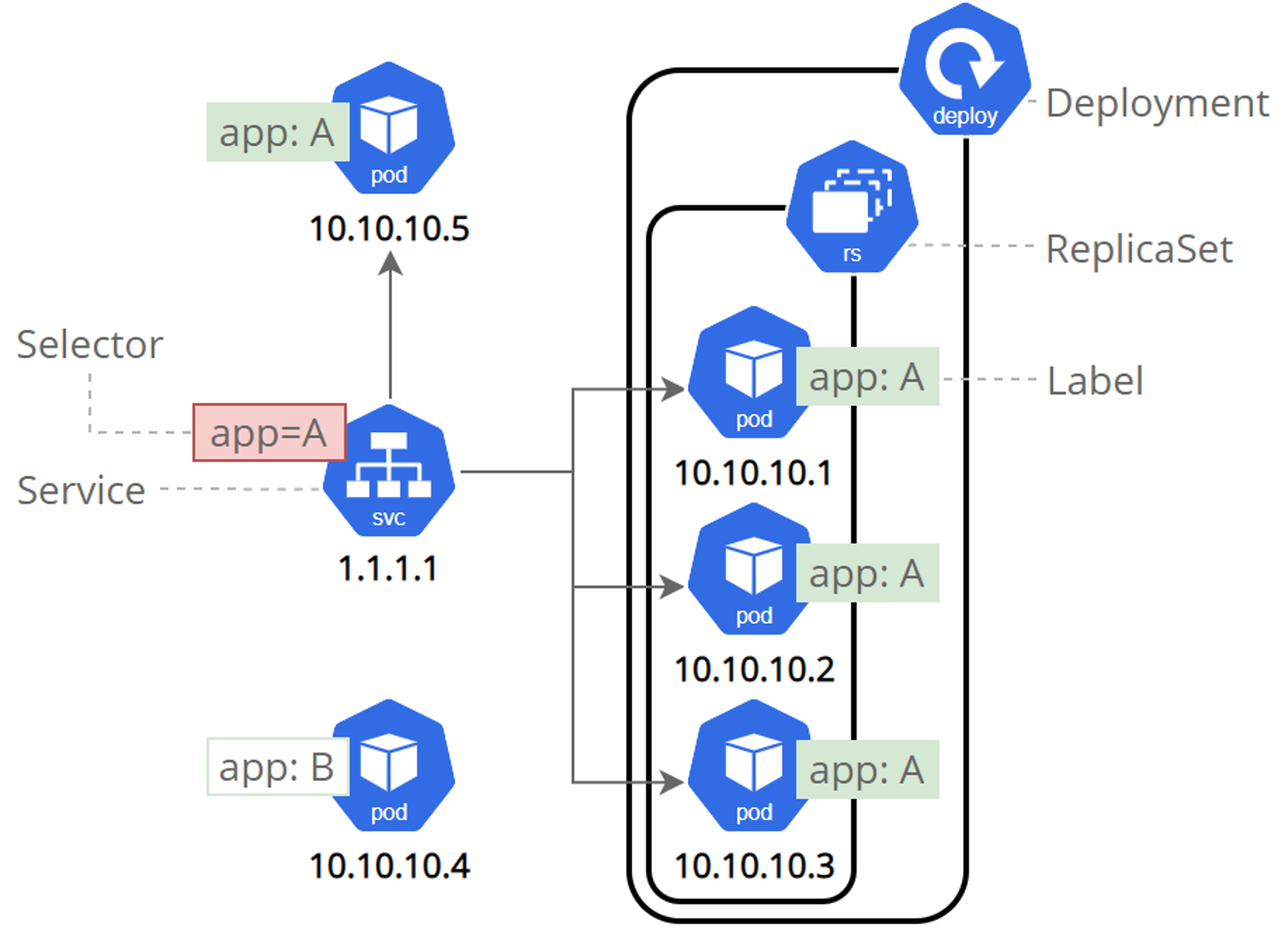
 Services can be exposed in different ways by specifying a type in the ServiceSpec:

* *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
* *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.
* *LoadBalancer* - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.
* *ExternalName* - Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up. This type requires v1.7 or higher of kube-dns, or CoreDNS version 0.0.8 or higher.

A Service routes traffic across a set of Pods. Services are the abstraction that allows pods to die and replicate in Kubernetes without impacting your application.

Labels are key/value pairs attached to objects and can be used in any number of ways:

* Designate objects for development, test, and production
* Embed version tags
* Classify an object using tags



To create a new service and expose it to external traffic we’ll use the expose command with NodePort as parameter.

kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080

To apply a new label we use the label command followed by the object type, object name and the new label:

kubectl label pods $POD\_NAME version=v1

Scaling

**Scaling** is accomplished by changing the number of replicas in a Deployment

To scale use kubectl scale deployments/kubernetes-bootcamp --replicas=2

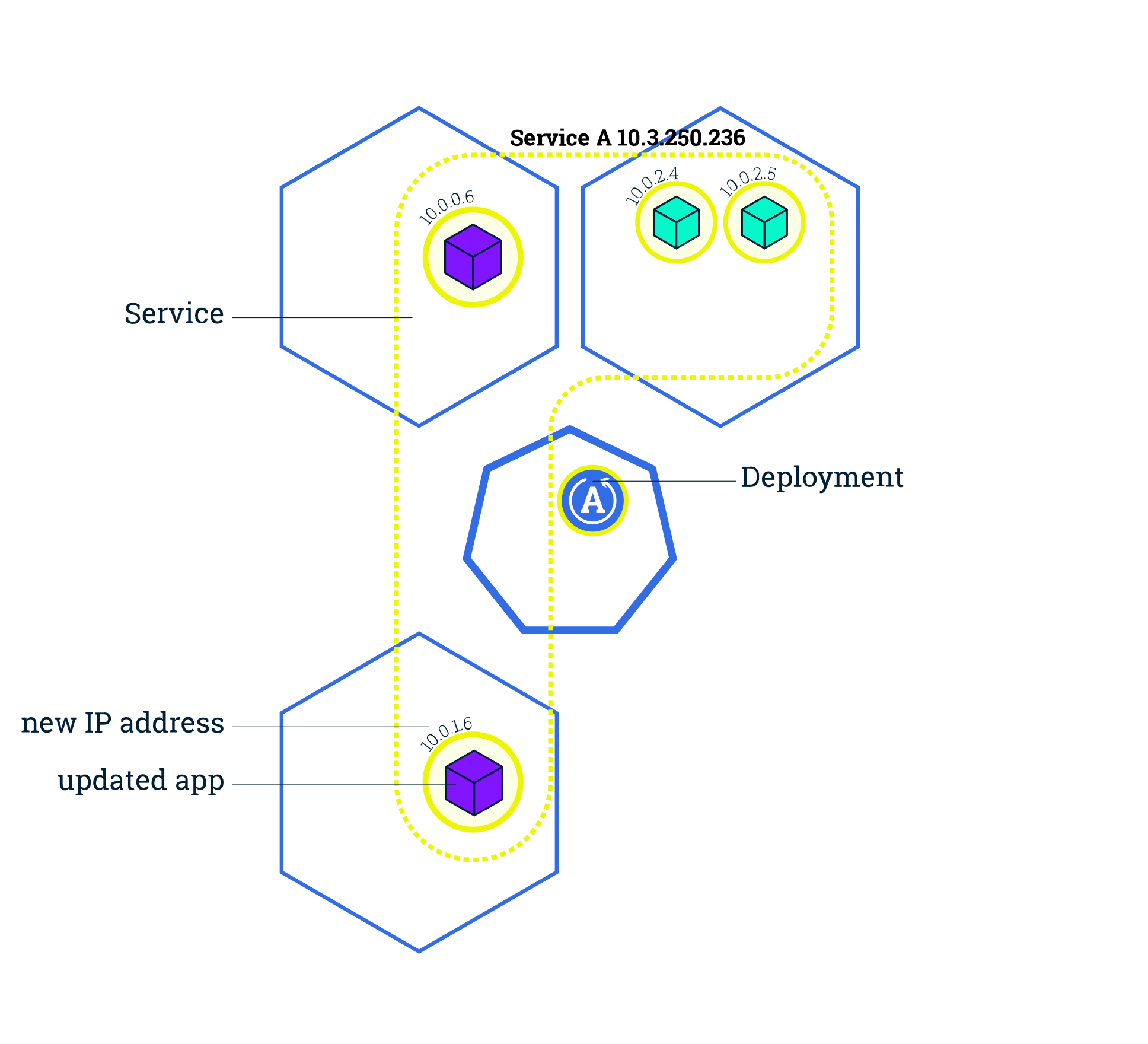
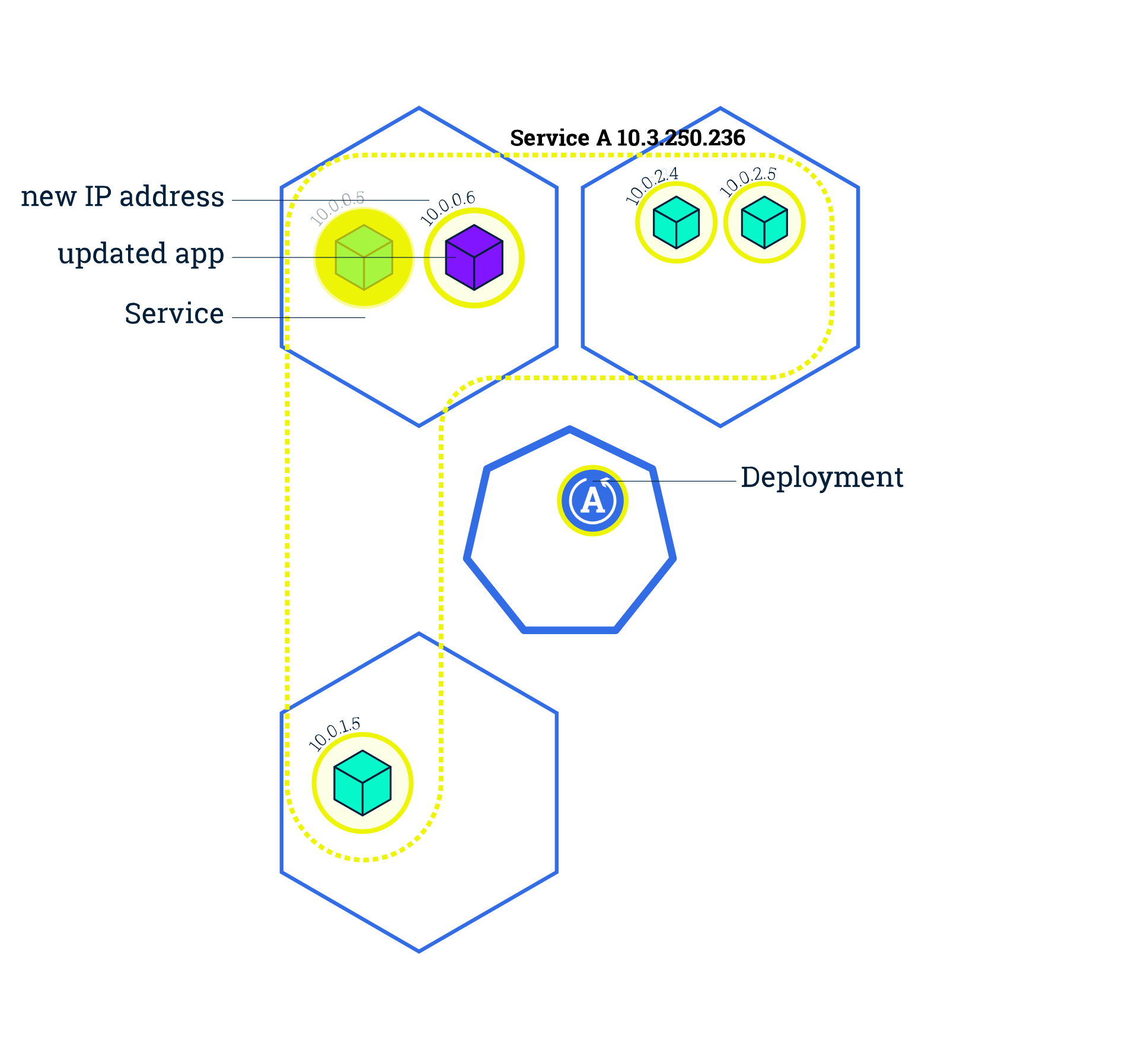
Rolling Updates

Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones.

Similar to application Scaling, if a Deployment is exposed publicly, the Service will load-balance the traffic only to available Pods during the update. An available Pod is an instance that is available to the users of the application.

Rolling updates allow the following actions:

* Promote an application from one environment to another (via container image updates)
* Rollback to previous versions
* Continuous Integration and Continuous Delivery of applications with zero downtime



To update the image of the application to version 2, use the set image command, followed by the deployment name and the new image version:

kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=jocatalin/kubernetes-bootcamp:v2